by Stefan Kalbermatter

DMX-512 Control

Build a USB-to-DMX-512 Converter

Stefan recently found himself in need of a controller for a lighting system. In this article, he describes how he used an FTDI FT232BM USB-to-serial converter chip to achieve DMX-512 control.

It was one of those times when someone has a project idea and you think, "Well, that's easy! Shouldn't be a problem!" Besides controlling an RFID reader via an RS-232, I was asked to control and dim eight different 230-V, 2-kW channels connected to halogen floodlights. The system had to be mobile, so the software had to run on a laptop.

As expected, the RFID reader portion, which is connected via RS-232, was relatively easy. Finding a suitable dimmer pack wasn't a problem either. They're relatively inexpensive. Decent music stores sell them for controlling stage light equipment via DMX-512.

I then started thinking about ways to connect the dimmer pack to my laptop. I knew that there are lots of DMX-512 controllers on the 'Net, but I couldn't find one to match my needs. I didn't want an expensive one. Furthermore, because of the laptop, I didn't want an ISA or LPT version. Finally, I wanted the entire control software to be self-made, so I needed something that I could easily control with my Windows application.

After searching the 'Net, I remembered the FTDI FT232BM USB-to-serial converter chip, which I had used a few weeks earlier for another project. Wouldn't it be possible to achieve DMX-512 control with this chip? Well, the answer is yes, and the result is presented in this article.

Before you begin this project, you must understand the DMX-512 protocol and the FT232BM chip. You also need to know how to generate a break pulse of at least 88 µs and a bit rate of 250 Kb. I'll cover all of these topics in addition to showing you how to drive the RS-485 signals and write the PC application.

DMX-512 PROTOCOL

The DMX-512 protocol is the most common communications standard used by lighting systems and related stage equipment. Data is transmitted at 250,000 bps via the RS-485 transmission standard over two wires plus ground. Normally, you will use a shielded 120-Ω, twisted-pair cable. The shield is used for connecting the ground pins.

DMX-512 provides up to 512 control channels per data link. Each channel was originally intended to control lamp dimmer levels. Think of it as up to 512 sliders on a lighting console connected to 512 lamps. Each slider's position is sent over the data link as 8-bit data having a value between zero and 255. The zero value corresponds to the lamp when it's completely off, while 255 corresponds to the lamp when it's on.

Today, most professional light systems (dimmer, scanner, moving lights, strobe, etc.) have a built-in DMX decoder. In theory, the USB-to-DMX-512 converter can control all these systems. Figure 1 shows the DMX-512's main characteristics.

DMX-512 BREAK PULSE

The break pulse must be at least 88 µs long. There are two possibilities for generating the pulse with the FT232BM chip.

The first option is to use 100,000 bps and send nine 0 bits. Because the start bit is already zero, just send one 0 byte to get a 90-µs break pulse. This solution is pretty slow because you need to change the bit rate twice for every DMX-512 stream that you send. You have to set it to 100,000 for the reset generation and then to 250,000 for the datastream transmission. Because of the USB transfers created by every bit rate change, this takes between 10 and 20 ms.

The second possibility involves using the RS-232 break command for the TXD line. At first I didn't like this solution because the length of the pulse, which isn't precisely generated by the hardware, depends more or less on the PC speed and the USB transfer time! But after some tests, it proved to be the faster solution. With a pulse duration of approximately 2 ms, this method is five to 10 times faster than the first. According to the DMX-512 specification, it must be at least 88 µs, so 2 ms is fine.[1]

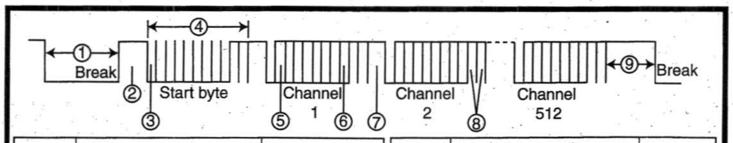


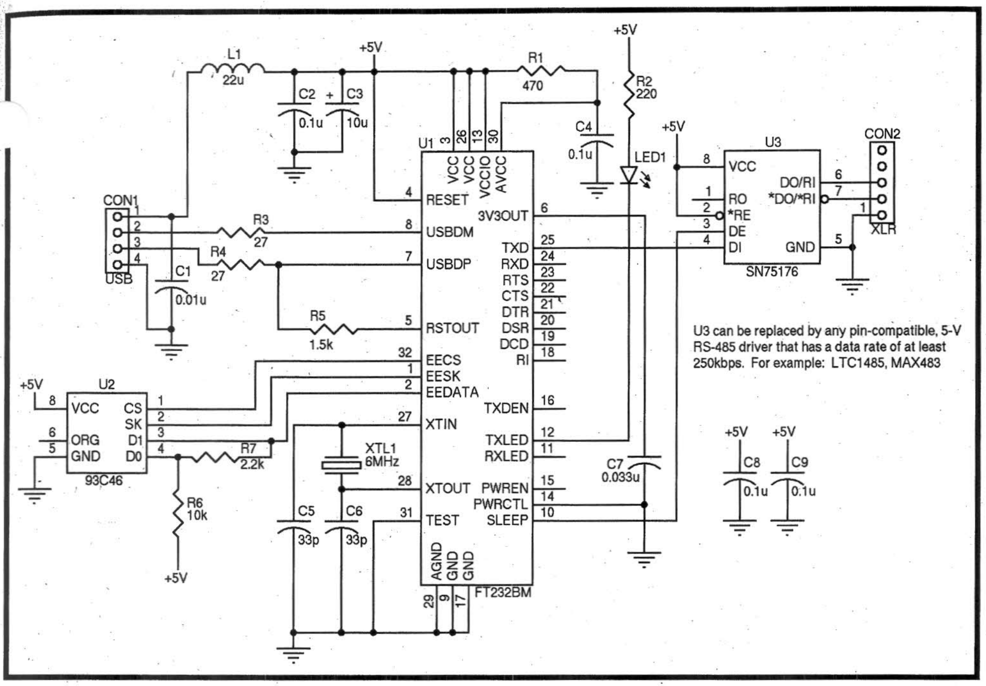
Figure	Description	Duration
1	Space for break (reset)	Minimum 88 µs
2	Mark after break (MAB)	8 µs–1s
3	Start bit	4 µs
4	Slot time	44 µs
5	Least significant data bit	4 µs

Figure	Description	Duration
6	Most significant data bit	4 µs
7	Mark time between slots	0–1s
8	Stop bit	4 µs
9	Mark before break (MBB)	0–1s

Figure 1—*The DMX-512 signal has an interesting structure. I included the table so you can study the duration of each section.*

DMX-512 DATASTREAM

Because of the universal bit rate generator inside the



U3 can be replaced by any pin-compatible, 5-V RS-485 driver that has a data rate of at least 250kbps. For example: LTC1485, MAX483

Figure 2—*It doesn't get any simpler than this. The design consists mainly of the FT232BM chip and an RS-485 line driver.*

FT232BM chip, it's easy to generate the necessary 250,000 bps when using the FTDI D2XX DLL USB drivers. In the past, the standard PC UART's bit rate limitation was seemingly one of the main problems for using it to generate a DMX-512 signal. Therefore, all existing DMX-512 interfaces normally have relied on an additional external microcontroller that generates the protocol timing.

Use the following data framing of the FT232BM chip to generate the correct datastream: 1 start bit, 8 data bits, no parity, and 2 stop bits. The start byte is sent first. In some special situations, alternative application-specific start bytes are defined. But this has to be set to zero for conventional DMX-512 devices.

After the start byte, send the data byte for channel 1 followed by channel 2 and so on. Theoretically, you only have to send as many channels as your configuration needs. For example, if you connect two six-channel dimmers, the first dimmer's DMX start address is set to one and the second dimmer's start address is set to seven. As a result, you only have to send the break pulse, start byte ($00), and 12 channel bytes. The Delphi sample software always sends all 512 channels, but you can easily adapt it to your needs.

HARDWARE

The hardware for this project mainly consists of the FT232BM, which was initially developed for building simple USB-to-RS-232 converters (see Figure 2).[2] The chip takes care of handling the USB protocol. It contains several additional intelligent features: a universal bit rate generator that can go up to 3 Mbps, RX FIFO, TX FIFO, and hardware handshaking.

Because of good documentation and the freely available universal PC drivers, the chip is popular not only for adding RS-232 ports to legacy-free PC Systems, but also for other devices that have to communicate with a PC or MAC via USB. Therefore, it's normally easy to find a chip distributor. On the other hand, if you don't feel like soldering the LQFP-32 yourself, just do what I did for the prototype: I bought a USB-to-RS-232 converter that's based on the FT232BM chip (sold by FTDI distributors like Saelig) and added the RS-485 driver chip and DMX-512 connector on the fly.

Because the RS-485 driver connects only to the power supply and two outputs of the USB chip, you can leave the other components on the original board untouched. As you can see in Photo 1a, I soldered four wires (VCC, GND, TXD, and –SLEEP) between the existing USB-to-RS-232 converter and the RS-485 chip. I connected the XLR connector to the RS-485 chip via three wires. It doesn't look too nice, but it works!

The –SLEEP pin connection is extremely important for complying with the USB specification, which explains that a USB device must consume less than 500 µA in USB Suspend mode. Without using the –SLEEP pin to disable the RS-485 driver in Suspend mode, this condition can't be satisfied.

I used standard DMX and USB cables (see Photo 1b). The red LED turns on as soon as the PC sends data.

DMX-512 CONNECTORS

Three different connector types are normally used for DMX-512 (see Table 1). As some designers have reported, the positive and negative are exchanged in some equipment (e.g., older Martin equipment). In case it doesn't work with yours, exchanging the positive and negative might do the trick. Note that Figure 2 shows the pinout for the XLR five-pin connector.

DMX-512 REFRESH RATE

Some DMX devices have a built-in timeout. If more than 1 s elapses without receiving a DMX signal, they automatically enter a predefined standby mode. Therefore, you need to send the DMX-512 stream periodically.

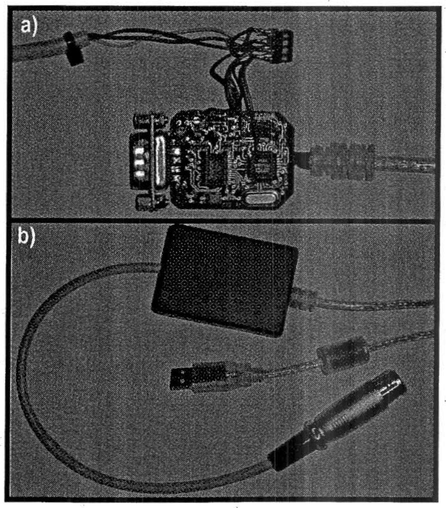


Photo 1a—*Take a look at the prototype converter. The PCB was taken from a standard USB-to-RS-232 converter using the FT232BM chip. The chip added on the fly was the RS-485 line driver SN75176.* **b**—*The prototype is in a case. By developing a custom PCB, you can drastically reduce the size of the case.*

By using all 512 channels and the minimum 88-µs reset time, the protocol allows you to refresh all the channels at 44 Hz. Tests have shown that because of break pulse generation and the USB latency, the maximum refresh rate achievable with this USB interface and the PC software is approximately 33 Hz. This changes slightly from one PC to another. You can easily test this by decreasing the TTimer object's interval.

Listing 1—*In the first section (*`RefreshBtnClick`*), I'm trying to connect to the USB device. If successful, I'll get and check the USB-to-DMX converter chip information. The* `OpenBtnClick` *routine opens the USB link and configures the USB device to generate the timings according to the DMX-512 specification. The transmission timer is enabled at the end.*

```
procedure TForm1.RefreshBtnClick(Sender: TObject);
// Check if there is an appropriate USB device. If so, display them.
var i : Integer;
    ok : boolean;
    DeviceCount : DWord;
    aDevice_String : Array [1..50] of Char;
    SerialNo    : String;
    Description : String;
begin
    // Get number of devices
  FT_GetNumDevices(DeviceCount,Nil,FT_LIST_NUMBER_ONLY);
  If DeviceCount > 0 then
  begin
    For I := 0 to DeviceCount-1 do
    Begin
         // Get the serial number
      ok := FT_ListDevices(i,aDevice_String,
              FT_OPEN_BY_SERIAL_NUMBER or
              FT_LIST_BY_INDEX) = FT_OK;
         // Store the serial number
      SerialNo := aDevice_String;
         // Get the device description
      if ok then ok := FT_ListDevices(i,aDevice_String,
                FT_OPEN_BY_DESCRIPTION or
                FT_LIST_BY_INDEX)) = FT_OK;
         // Store the device description
      Description := aDevice_String;
      if ok then
      begin
         // Display the serial number and the description of this
device.
      end;
     End;
    end;
end;

procedure TForm1.OpenBtnClick(Sender: TObject);
// Open the selected USB device.
Var Index : Integer;
begin
  Index := SelectedDevice;
  If FT_Open(Index,FT_Handle) = FT_OK then
  Begin
        // Reset the chip.
        FT_ResetDevice(FT_Handle);
        // Define timeouts. (RX timeout is not important for this
        // application.)
        FT_SetTimeouts(FT_Handle,16,50);
        // Set the DMX data rate.
        FT_SetBaudRate(FT_Handle,250000);
        // Set DMX-512 data characteristics.
        FT_SetDataCharacteristics(FT_Handle,FT_DATA_8,
               FT_STOP_2,FT_PARITY_NONE);
        // Disable handshaking.
        FT_SetFlowControl(FT_Handle,FT_FLOW_NONE,FT_XON_Value,
        FT_XOFF_Value);
        // Restart the Ttimer object.
        Timer1.Enabled := True;
  end;
end;
```

SOFTWARE

I wrote the PC software in Delphi 5. However, because you don't need fancy components, you should be able to compile it with any Borland Delphi version (between 2 and 7). The entire program control is located in the main.pas file, which is posted with the other source files on the *Circuit Cellar* ftp site. The file D2xxUnit.pas is an object Pascal wrapper for ftd2xx.dll, which is a slightly modified and extended version of the one supplied by FTDI.

If you're unfamiliar with programming in Borland Delphi, you'll probably be surprised by the software's simplicity. It mainly consists of initialization, an event handler—which is called every time you move one of the 12 sliders—and a TTimer event handler that sends the data in a predefined period to the USB device.

Let me reiterate that the Delphi code is extremely simple. With that said, let's focus on the USB-related portion of the four most important routines. I removed the GUI-related stuff from Listings 1 and 2.

USB DRIVERS

You may be wondering whether or not you need to write device drivers. Don't worry; it isn't necessary. FTDI offers two sets of USB drivers. On one side is the virtual COM port (VCP) driver, which installs a VCP. This is advantageous because it can be accessed like a normal serial COM port, but it isn't suited too well for this project because the driver by default supports only the Windows standard RS-232 bit rates, which normally only go up to 115,200 bps. The FTDI web site describes a possible workaround for this problem, but it seems to be tricky and not the best option.

The second driver is the D2XX DLL USB driver, which provides direct access to the chip via the supplied DLL/driver combination. This does away with the hassles of legacy COM port interfaces. It's advantageous because you can easily program any bit rate including the required 250,000 bps.

USB VID & PID

Every USB device needs specific vendor ID (VID) and product ID (PID) numbers programmed into its EEPROM. As soon as you connect the device, the operating system will use the two numbers to locate the appropriate drivers.

If the FT232 chip discovers an empty EEPROM connected to the EECS, EESK, and EEDATA pins, it automatically uses the FTDI VID 0x0403 and the default PID 0x6001. The standard USB-to-RS-232 cable I bought for my prototype was also programmed to this FTDI default VID/PID combination.

The USB-to-DMX-512 interface should work with the default VID/PID combination, as long as you supply the correct direct drivers during installation. But, unfortunately, this could lead to driver mismatch with other USB devices containing the same chip. Therefore, you have to reprogram the EEPROM with the PID 0xEC70, which has been reserved for the USB-to-DMX-512 project.

There are different ways to reprogram the EEPROM. I'll focus on one option. At first glance you'll think it's a bit complicated. But after you consider the different problems you could encounter with the different Windows versions (especially XP), it's the easiest solution.

DMX-512 signal	XLR 3 pin	XLR 5 pin	Stereo Jack 6.5 mm
Ground (shield)	1	1	Sleeve
Data –	2	2	Ring
Data +	3	3	Tip
Second Data – (optional)		4	
Second Data + (optional)		5	

Table 1—*Here are the pinouts for the most commonly used DMX-512 connectors.*

INSTALLATION

When you connect the device for the first time, Windows asks for the corresponding device drivers. If you don't have the original disks, use the necessary drivers in the VCP driver subfolder posted on the *Circuit Cellar* ftp site. Windows XP will automatically install them.

Next, you must remove the USB devices that might contain an FTDI chip using the default VID/PID combination. If you're not sure about this, remove all of the USB devices except the one you want to reprogram.

Now you can start the USB2DMX512-Reprogrammer.exe program, which is posted on the *Circuit Cellar* ftp site. After start-up, the status line should be blank. If you see the "Please connect the device!" message, it means your USB device isn't connected, it isn't working properly, or that there is a problem with the driver installation.

Next, press the "Try to reprogram now!" button. If everything works, the status line reads, "Device reprogrammed with USB2DMX512 PID." It doesn't just reprogram the PID; it also programs the device name and the maximum allowable current (90 mA) needed from the USB port.

After disconnecting and reconnecting (wait several seconds in between) your device, it will be recognized by Windows as "USB2DMX512 Converter." Windows will then ask for drivers. At this point you can install the final drivers, which are located in the USB2DMX512 drivers subfolder. That's all! Now you can run the USB2DMX512.exe file.

If your USB device is properly installed, you should see it listed in the top section of the window (see Photo 2). Select it and click the Open button. If you don't receive an error message, you're properly connected to the device. Now you can move the 12 sliders as you see fit. By setting the start channel to 25, for example, channel 1's slider becomes channel 25, channel 2's becomes channel 26, and so on.

The aforementioned program is an easy sample. Now it's up to your imagination. Think about some preset buttons and hotkey presets. By adding TCP/IP functionality to your Delphi application, you can also control your light from any PC via your wireless LAN by connecting a dimmer pack with this interface to your server.

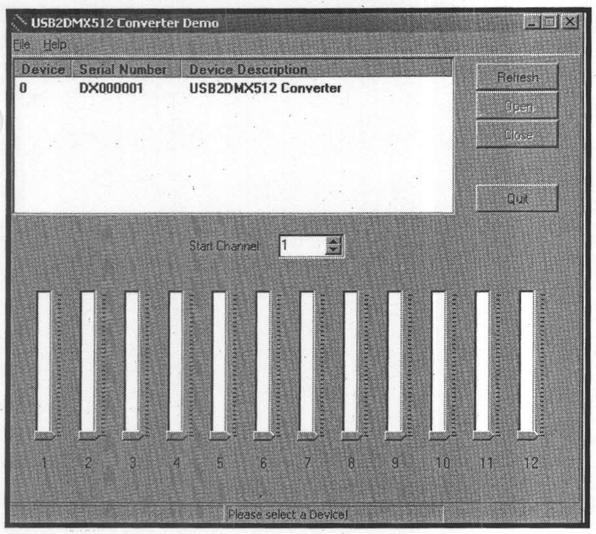
POSSIBLE IMPROVEMENTS

My project works well, but you can make several improvements. For instance, you can put the `send` routine in a separate Windows thread. With this, the DMX-512 transmission would become more independent of the application. If you perform a time-consuming task in the main application, the DMX-512 refresh can be suspended for some time.

Galvanic isolation is another option.

Listing 2—*The `Timer1Timer` procedure is called periodically by the transmission timer. It generates the DMX-512 reset pulse and sends the 512 channels to the chip. `CloseBtnClick` stops the transmission by disabling the transmission timer and closing the USB link.*

```
procedure TForm1.Timer1Timer(Sender: TObject);
// Send the DMX stream, triggered by a Ttimer component.
var ok : boolean;
     Count : DWord;
Begin
 // Stop timer.
 Timer1.Enabled := False;
 // Generate the DMX break.
 ok := FT_SetBreakOn(FT_Handle)=FT_ok;
 if ok then ok := (FT_SetBreakOff(FT_Handle) = FT_OK);
 // Send the start byte and the 512 DMX-512 channels.
 if ok then ok := (FT_Write(FT_Handle,@DMX_Buf,
                   DMXChannelsMax+1,Count)=FT_ok);
 // If we get an error, we stop the transmission, otherwise
     restart.
 if ok then Timer1.Enabled := True
       else CloseBtnClick(Self);
end;
procedure TForm1.CloseBtnClick(Sender: TObject);
// Close the USB device.
Begin
 // Stop the DMX refresh.
    Timer1.Enabled := False;
 // Close the USB device.
 FT_Close(FT_Handle);
end;
```



Device	Serial Number	Device Description
0	DX000001	USB2DMX512 Converter

Photo 2—*This simple sample application was written in Delphi 5. The top part lists the available USB-to-DMX-512 devices. The lower part allows you to set the 12 channels. Feel free to adapt and extend it to fullfill your needs.*

If you use the DMX-512 interface on stage, the galvanic isolation of the line driver could improve reliability and protect your computer against possible over-voltage on the DMX-512 cables.

DMX-512 TERMINATION

The DMX-512's data rate is fast enough to be put in the "radio frequency" category of signals. Therefore, you need proper termination. The simple rule is this: at the end of each chain of DMX-512 receivers, there should be a 120-Ω termination resistor between Data + and Data –. For short cables, it works without termination, but it's cleaner to add this termination anyway.

REAL-LIFE TESTS

I've used the circuit with several different brands of show scanners and dimmer packs. It worked properly with all of these devices without a blackout or any problems. Therefore, you can be sure that it will also work with other DMX-512-compatible lighting equipment, such as a huge 10-kW Space Cannon skylight fixture. ■

Stefan Kalbermatter has worked in Switzerland for more than eight years as an independent hardware and software consultant. You may contact him via his web page www.kalbermatter.ch.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2004/170.

REFERENCES

[1] USITT, "Entertainment Technology—USITT DMX512-A Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories Rev. 3," Draft BSR E1.11.

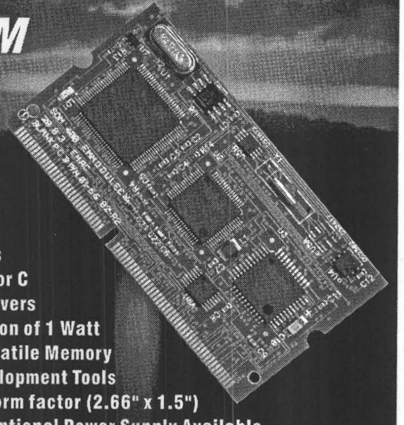
[2] FTDI, "FT232BM USB UART (USB—Serial) I.C.," DS232B, V. 1.4, 2004, www.ftdichip.com/FTProduct.htm.

SOURCE

FT232BM USB UART IC
FTDI, Ltd.
www.ftdichip.com

TINI® Webserver
Internet Appliance Engine

SOM-400EM



- DS80C400 Processor
- IPv6 Fast Ethernet
- 3 Serial Ports & SPI
- SODIMM Bus Expansion
- Uses a bash shell subset
- 1 MB Flash & 2 MB RAM
- CAN 2.0B & 1-Wire® Ports
- Programmable in Java™ or C
- Telnet, FTP, and HTTP Servers
- Typical Power Consumption of 1 Watt
- Real Time Clock & Nonvolatile Memory
- Robust, Free Java™ Development Tools
- Small, 144 pin SoDIMM form factor (2.66" x 1.5")
- Carrier/Socket Board & Optional Power Supply Available

The SOM-400EM is a System on a Module, based on the Dallas® Tiny InterNet Interface (TINI®) Processor. This 8051 code-compatible processor makes it extremely easy to create a smart Network / Internet capable device. The SOM comes complete with Embedded Java OS and File System. Write sophisticated network applications in days instead of months. Single unit pricing starts at $100. For additional information visit, http://www.emacinc.com/som/.

Since 1985
OVER
19
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

Phone: (618) 529-4525 • Fax: (618) 457-0110 • Web: www.emacinc.com

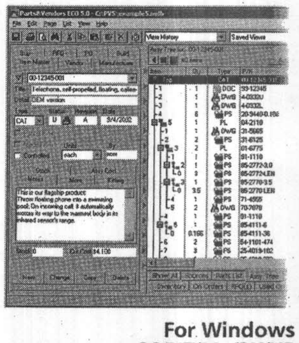
---

Parts List Software
for Engineers and Designers

- Easily create and manage multi-level parts lists for products in development...and after.
- Track sources for items with multiple price breaks.
- Calculate product costs at any quantity.
- Launch CAD, viewer or browser from any item.
- Automatically generate RFQs or POs.

New Version 5.0

- **New** Report Layout Editor customizes reports/labels.
- **New** Connection to QuickBooks 2002/2003 Pro simplifies accounting (US version only).
- **New** Multi-currency for foreign suppliers eases exchange rate calculations.



Parts& Vendors™

For Windows 98/NT/Me/2K/XP
3 Editions, starting at $99 + s/h

Visit www.trilogydesign.com and download our FREE DEMO.

Trilogy DESIGN™

Or, Call 800-280-5176
530-273-1985 Fax 530-477-9106
P.O. Box 2270, Grass Valley, CA 95945